Chapitre 1 :
Prise en main de Python

I Utilisation de Python

1.1 Préambule

Python est un langage de programmation dont la première version, développée par Guido van Rossum, est apparue en 1989 et qui a continué à être développé jusqu'à la version 3 disponible actuellement. C'est un langage informatique largement utilisé qui présente l'avantage d'être sous licence libre et d'être gratuit.

Un environnement de développement intégré (IDE : Integrated Development Environment) est un ensemble d'outils permettant de rationaliser la création et le développement de programmes.
Il existe différents environnements de développement gratuits et compatibles avec Python : Idle, Spyder, Pyzo ... Nous utiliserons l'environnement **Pyzo** qui a été utilisé les années précédentes lors des oraux du concours Centrale-Supélec.
Il comprend entre autre :

- un éditeur permettant d'écrire et de sauvegarder des programmes longs ;
- une console interactive (shell) qui permet d'interagir avec Python directement.
- une aide interactive

1.2 Installation

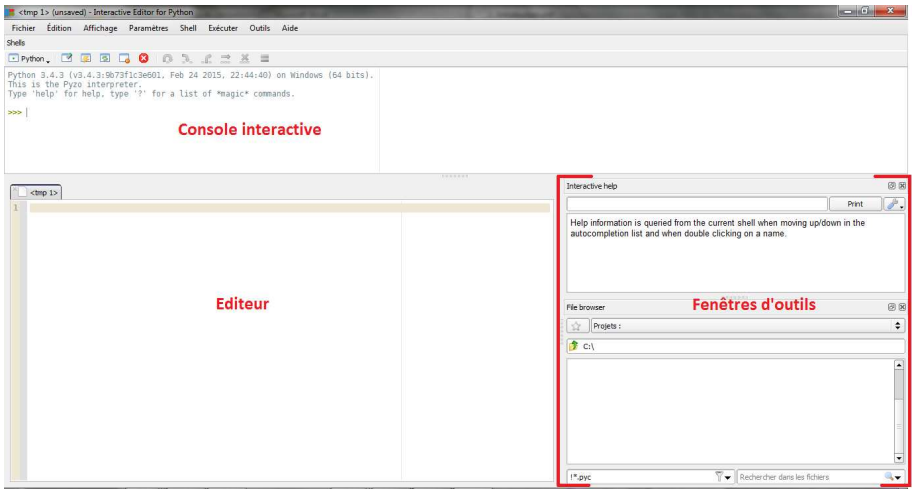
• Il faut commencer par télécharger **Pyzo** à l'adresse suivante :

http ://www.pyzo.org/start.html

et l'installer. Pyzo est disponible pour Windows, Linux et macOS. Vous venez alors d'installer l'éditeur de l'environnement Pyzo.
• Puis, il vous faut ensuite installer **anaconda** (comprenant entre autre un interpréteur Python mais également de nombreux packages scientifiques) un lien est disponible sur la même page. Choisissez la version 3 de Python. Il est conseillé de l'installer à l'endroit par défaut.
• Lancez Pyzo. L'environnement anaconda est généralement détecté automatiquement. Il vous suffit alors d'accepter de l'utiliser.
• Vous pouvez enfin si vous le souhaitez mettre Pyzo en français. Pour ce faire, lancez Pyzo, allez dans le menu *Settings*, *select language* et choisissez *French*. Il vous faut redémarrer Pyzo pour que cette modification soit prise en compte.

1.3 Lancement de Pyzo

Au lancement de Pyzo, vous obtenez un écran de la forme :



La console ou shell :

La console interactive s'utilise comme une calculatrice.

La validation se fait en appuyant sur entrée. Chaque ligne tapée est alors immédiatement exécutée.

Par souci de lisibilité, la console n'est adaptée qu'à des calculs ou des programmes courts et qui n'ont pas besoin d'être sauvegardés. Les programmes plus longs ou destinés à être sauvegardés sont écrits dans l'éditeur.

Dans la console interactive, on ne peut pas modifier une ligne de commande précédemment exécutée. On pourra cependant rappeler au niveau de l'invite de commande une des lignes précédemment exécutées grâce aux flèches de défilement. On pourra alors la modifier avant de relancer le calcul.

Dans tout le cours, le symbole >>> signifie que ce qui suit a été entré dans la console. Ce symbole ne doit pas être tapé.

Éditeur

L'éditeur nous permet d'écrire et de sauvegarder des programmes.

Le programme doit être exécuté. Pour cela, on peut aller dans le menu *exécuter* puis de sélectionner une commande d'exécution ou utiliser un des raccourcis suivants :

- Ctrl + E : exécution du fichier,
- Alt+Return : exécution de la sélection.

Les instructions sont alors lues et exécutées, le résultat, lorsqu'il y en a un, s'affiche dans la console interactive.

Pour créer un nouveau fichier, on utilise le menu fichier, Nouveau. On peut alors taper nos instructions, les sauvegarder et les exécuter.

Créer un fichier pour sauvegarder le travail de chaque séance.

Les fenêtres d'outils

Il est possible de choisir les fenêtres d'outils qui s'affichent en allant dans le menu Outils. On peut notamment ouvrir la fenêtre d'aide interactive qui permet d'obtenir la documentation sur les commandes utilisées au fur et à mesure de leur saisie.

II Variables

2.1 Généralités

Une variable permet de stocker des données pour les réutiliser ensuite.

La création d'une variable signifie que l'on réserve une case mémoire dans laquelle la valeur de la variable est stockée. On pourra alors accéder à cet emplacement mémoire pour accéder ou modifier la valeur de la variable.

Chaque variable doit avoir un nom. Ce nom peut être formé de lettres majuscules ou minuscules, de chiffres et du caractère _. Le nom ne peut pas être un mot réservé du langage (par exemple `for`, ...).

Choisir un nom de variable qui a un sens.

2.2 Déclaration et affectation

Lorsque l'on crée une variable, on dit qu'on la déclare. En Python, pour déclarer une variable, il faut toujours lui donner une valeur. Le symbole = permet de déclarer une variable. Par exemple, la commande `a=1` permet de déclarer la variable `a` et initialise sa valeur à 1. Cette commande ne produit pas d'affichage.

```
>>> a=1
>>> a
1
```

Il faut faire attention à l'ordre :

- le membre de gauche est le nom de la variable qui va être modifiée,
- le membre de droite est la valeur qui va être affectée à la variable.

Ainsi : 1=a n'a pas de sens.

L'affectation n'est pas commutative.

Une fois que la variable est créée, on peut changer sa valeur en utilisant toujours l'instruction =, il s'agit alors d'une affectation.

```
>>> a=1
>>> a=2
>>> a
2
>>> a=a+3
>>> a
5
```

On peut également faire des affectations multiples, en utilisant une virgule. Il ne faut pas confondre la virgule et le point virgule qui sert à séparer deux instructions

```
>>> a,b=4,5
>>> a
4
>>> b
5
>>> a,b
(4,5)
>>> a ;b
4
5
```

En Python, on peut utiliser des raccourcis (qui correspondent à une mise à jour de l'ancienne variable et pas à une affection) qui sont les suivants :

- a+=b pour a=a+b,
- a-=b pour a=a-b,
- a*=b pour a=a*b,
- a/=b pour a=a/b.

```
>>> x=2
>>> x+=3
>>> x
5
>>> x*=x
>>> x
25
```

Exercice 1

Donner les réponses aux commandes suivantes :

```
>>> a=2
>>> b=a-3
>>> a=a+2
>>> b
```

.....

```
>>> x=5
>>> x+=x
>>> x*=x
>>> x
```

.....

```
>>> x,y=5,2
>>> x+=y
>>> y+=x
>>> x,y
```

.....

Exercice 2

Echange du contenu de deux variables

On considère deux variables x et y qui ont comme valeurs respectives a et b. On souhaite échanger les contenus de ces variables, c'est-à-dire avoir $x = b$ et $y = a$.

On choisit arbitrairement des valeurs de a et b. (distinctes bien-sûr!)

1. Proposer une solution faisant apparaître une troisième variable z.

   .....

   .....

   .....

   .....

   .....

2. Proposer une solution utilisant une affectation multiple.

   .....

   .....

   .....

3. Etudier et exécuter la commande suivante :

```
>>> x=x+y
>>> y=x-y
>>> x=x-y
>>> x
>>> y
```

   .....

   .....

   .....

   .....

   .....

III Types

En Python, les variables sont typées, c'est-à-dire qu'elles ont un type qui correspond à l'objet représenté. Lors de la déclaration d'une variable, le type n'est pas précisé, il est automatiquement transmis par le membre de droite à la variable. On parle de typage dynamique. Le type d'une variable n'est pas fixe, il peut être modifié lorsqu'on affecte une nouvelle valeur à la variable.

Pour accéder au type d'une variable, on utilise la commande `type`.

3.1 Types de base

Les principaux types de base sont les suivants :

1. Les entiers : leur type est `int`. En Python, il n'y a pas de limite dans le nombre de chiffres apparaissant dans l'écriture des entiers. Le résultat obtenu est donc exact.
   Les opérations sur les entiers sont :
   - addition `+`, soustraction `-`, multiplication `*`,
   - puissance `**`,
   - quotient de la division euclidienne de deux entiers `//`, reste de la division euclidienne de deux entiers `%`.

2. Les flottants : leur type est float. La virgule décimale est notée par un point. En Python, les flottants sont stockés en utilisant un développement binaire tronqué. Le résultat obtenu n'est donc pas exact.
   Les opérations sur les flottants sont :
   - addition +, soustraction -, multiplication *, division /,
   - puissance **.

Comme en mathématiques, les règles de priorité des opérations sont respectées. Il faut donc bien indiquer les parenthèses. En cas d'oubli d'une parenthèse fermante, on obtient des points de suspension et pas de nouvelle invite de commande :

```
>>> x=4*(2+3
...
```

```
>>> x=4*(2+3
...)
>>>
```

On peut faire des opérations sur des variables de types entier et flottant, la règle du typage fort est alors appliquée : les variables entières sont converties automatiquement en flottants.

3. Les booléens : leur type est bool. Les booléens servent à évaluer des expressions logiques, leurs valeurs possibles sont True et False.
   Les opérations sur les entiers sont :
   - négation not,
   - ou logique or,
   - et logique and.

Les comparaisons : <, >, <=, >=, == (égal), != (différent) donnent des résultats booléens.
L'exemple suivant montre, en utilisant des booléens, que la représentation des flottants n'est pas exacte.

```
>>> 0.1+0.1+0.1==0.3
False
>>> 0.5+0.5+0.5==1.5
True
```

On n'utilisera pas de test d'égalité entre flottants.

L'évaluation des booléens est dite "paresseuse", cela signifie que :

- dans l'évaluation de :

$$b_1 \text{ or } b_2 \text{ or } b_3 \text{ or } \ldots \text{ or } b_n,$$

  b_1 est testée, puis b_2, ...et dès qu'une des valeurs est True, les autres ne sont plus testées et la réponse est True,
- dans l'évaluation de :

$$b_1 \text{ and } b_2 \text{ and } b_3 \text{ and } \ldots \text{ and } b_n,$$

  b_1 est testée, puis b_2, ...et dès qu'une des valeurs est False, les autres ne sont plus testées et la réponse est False.

Cela implique des problèmes de non commutativité.
On aura, par exemple, les résultats suivants :

```
>>> 1/3<1/2 or 1/2<1/0
True
>>> 1/2<1/0 or 1/3 <1/2
Traceback (most recent call last) :
    File "<stdin>", line 1, in <module>
ZeroDivisionError : division by zero
```

Bien lire les messages d'erreur.

Exercice 3

Donner les résultats aux commandes suivantes :

```
>>> a=1+2
>>> type(a)
.....
>>> b=1+2.
>>> type(b)
.....
>>> c=1/2
>>> type(c)
.....
>>> d=4/2
>>> type(d)
.....
>>> e=25**(1/2)
>>> type(e)
.....
```

```
>>> a=3
>>> b=4.5
>>> a==b
.....
>>> a<=b
.....
>>> a !=b
.....
>>> a%2==0
.....
>>> (a+2<=b) or (a-2<=b)
.....
>>> (a+2<=b) and (a-2<=b)
.....
```

3.2 Types structurés

Les types structurés seront étudiés dans le prochain chapitre. Il s'agit des tuples, des listes, des chaines de caractères et des dictionnaires.

IV Modules

4.1 Importation de modules

Les modules (ou bibliothèques) regroupent des fonctions et des constantes utilisables uniquement lorsque le module a été importé.
Il existe plusieurs façons d'importer un module. On appellera module le module à importer :

- import module : tout le contenu de module est importé mais il faudra préciser à chaque fois la provenance en utilisant un préfixe. Par exemple, un élément f de module sera utilisé avec la commande module.f,
- import module as mod : tout le contenu de module est importé et le nom du module est abrégé en mod. Il faudra toujours préciser à chaque fois la provenance en utilisant un préfixe, mais abrégé. Par exemple, un élément f de module sera utilisé avec la commande mod.f,
- from module import * : tout le contenu de module est importé et il est inutile de préciser la provenance de ses éléments. Par exemple, un élément f de module sera utilisé avec la commande f,
- from module import f_1, f_2 : seuls les objets f_1 et f_2 de module sont importés et il est inutile de préciser leur provenance. Par exemple, un élément f_1 de module sera utilisé avec la commande f_1.

La commande dir(module) donne la liste des éléments de module, après son importation.

Importer les modules une seule fois, en début de fichier.

4.2 Le module math

Les fonctions mathématiques usuelles ne sont pas présentes par défaut :

```
>>> cos(0)
Traceback (most recent call last) :
File "<stdin>", line 1, in <module>
NameError : name 'cos' is not defined
```

Le module contenant les fonctions mathématiques usuelles est le module math, on dispose, entre autres, de :

- cos : cosinus,
- sin : sinus,
- tan : tangente,
- pi : nombre π,
- exp : exponentielle,
- e : nombre e,
- log : logarithme népérien ln,
- log10 : logarithme décimal,
- sqrt : racine carrée.

Exercice 4

Calculs dans $\mathbb{R}$

Afficher les valeurs suivantes de :

1. $\sqrt{2+\sqrt{2}}$,
2. $\dfrac{\sin(10^{-5})}{10^{-5}}$,
3. $2(1+\ln(e+1))$
4. $\sqrt[3]{27}+2$,
5. $\cos\left(\frac{\pi}{4}\right)$.

V Entrées et sorties

Les entrées et les sorties sont les échanges d'information entre l'utilisateur et le programme.

5.1 La commande print

Pour afficher la valeur d'une variable, on utilisera la commande print.
Pour afficher une phrase contenant un ou des résultats, on utilisera toujours la commande print mais en utilisant des chaînes de caractères (délimitées par des guillemets ou des apostrophes) ainsi que des variables.

```
>>> a=5
>>> print(a)
5
>>> print("La valeur de a est",a)
La valeur de a est 5
>>> b=3
>>> print("La valeur de a est",a, ", celle de b est",b)
La valeur de a est 5, celle de b est 3
```

5.2 La commande input

Si on souhaite faire un affichage qui demande une réponse de l'utilisateur, on utilisera la commande input.

La commande input permet de stocker la valeur rentrée par l'utilisateur. Le programme attend donc une réponse de l'utilisateur.

```
>>> a=input('Que vaut 2+2 ?')
Que vaut 2+2 ? |
```

Le curseur est placé à la fin de la dernière ligne et indique l'attente d'une réponse.

```
>>> a=input('Que vaut 2+2 ?')
Que vaut 2+2 ? 4
>>>
```

L'utilisateur a répondu 4. Cette valeur est stockée dans a.

```
>>> a
'4'
```

La réponse à la commande input est une chaîne de caractères, si on veut utiliser cette réponse comme un nombre, il faut utiliser la commande eval.

Exercice 5

Donner les résultats aux commandes suivantes :

```
>>> a=input('Quel est ton age ?')
.....
>>> a
.....
>>> a+1
.....
>>> a=eval(input('Quel est ton age ?'))
.....
>>> a
.....
>>> a+1
.....
```

VI Ecriture de fonctions

6.1 Structure

Une fonction est un bloc d'instructions qui a reçu un nom, dont le fonctionnement dépend de paramètres et qui peut renvoyer un résultat.
La structure est la suivante :

```
def nom(paramètres):
    instructions
    return résultat
```

Lorsqu'on exécute une fonction, par exemple f(x), la valeur de x est d'abord calculée puis la fonction f est exécutée avec la valeur calculée.

Les fonctions seront rentrées dans l'éditeur et pas dans la console et seront obligatoirement exécutées avant d'être appliquées.

Une fonction doit être exécutée après chaque modification et testée sur un ou des exemples.

L'indentation (le bloc de quatre espaces précédent les instructions) est obligatoire : cela signifie que l'on se situe à l'intérieur de la fonction.

On peut utiliser la touche Tab ⇤⇥ pour obtenir une indentation et Maj+Tab pour enlever une indentation.
La fin de l'indentation, signifie aussi la fin de la fonction.
La commande `return` donne la valeur du résultat et termine la fonction. Elle est facultative.

Les commandes qui suivent un return ne sont pas lues (code mort).

Il ne faut pas confondre `return` et `print` :

- `return` donne la valeur qui est prise par la fonction
- `print` affiche une valeur quelconque.

Voici quelques exemples :

```
def f1(x) :
    print(x**2)
def f2(x) :
    return x**2
```

Ce fichier est entré dans l'éditeur puis exécuté. Pour obtenir les résultats qui suivent, on utilise la console.

➪ **Exemple 1 :**

```
>>> f1(5)
25
>>> y=f1(5)
25
>>> y
>>> f2(5)
25
>>> z=f2(5)
>>> z
25
```

Dans cet exemple, il n'y a pas de message d'erreur lorsqu'on demande y ce qui serait le cas si la variable y n'était pas définie. y est donc bien définie mais n'a pas de valeur.

➪ **Exemple 2 :**

On considère la fonction suivante :

```
def f3(x) :
    print(x**2)
    return x
    print('x')
```

On a :

```
>>> f3(5)
25
5
>>> y=f3(5)
25
>>> y
5
```

Les caractères précédés du caractère # sont des commentaires, ils ne sont pas exécutés. Les commentaires sont importants car ils permettent d'expliquer la fonction écrite.

Une fonction doit être commentée.

Les erreurs de syntaxe sont repérées lors de l'exécution de la fonction mais les autres erreurs sont repérées lors de l'appel.

➪ **Exemple 3 : Erreurs classiques**

```
def f(x) :
    x=1/(x-x)
    return x
>>> f(2)
...
x=1/(x-x)
ZeroDivisionError : division by zero
```

Lorsqu'on divise par 0, il n'y a pas d'erreur lors de l'exécution mais uniquement lors de l'appel, c'est pourquoi il est important de tester les fonctions sur des exemples.

```
def f(x) :
    x=3*(x+5
    x=2*x
    return x
>>>
...
x=2*x
  ^
SyntaxError : invalid syntax
```

L'erreur de syntaxe est l'oubli de la parenthèse. Cependant, l'erreur est signalée au début de la ligne suivante car c'est à ce moment qu'elle a été vue. On pense donc à regarder la ligne précédente.

```
def f(x) :
     x=3*(x+5)
    x=2*x
    return x
>>>
...
x=2*x
  ^
IndentationError : unindent does not match any outer indentation level
```

Le problème est l'espace en trop à la ligne précédent `x=2*x`.

```
def f(x) :
    x=2(x+1)
    return x
>>> f(2)
...
x=2(x+1)
TypeError : 'int' object is not callable
```

On a écrit `x=2(x+1)` or 2 est un entier et les parenthèses correspondent à l'appel de fonction. L'écriture correcte est :

```
def f(x) :
    x=2*(x+1)
    return x
>>> f(2)
6
```

Exercice 6

Ecriture de fonctions

1. Ecrire une fonction qui prend comme argument un flottant x et qui renvoie x^2 et x^3.
2. Ecrire une fonction qui prend comme argument un flottant x et qui renvoie le booléen True si $x \geq 1$ et False sinon.
3. Ecrire une fonction qui prend comme argument un entier n et qui affiche la valeur de n est . . .et on a n+2=. . ..

6.2 Portée des variables

Il existe deux types de variables :

- Les variables locales sont les variables définies à l'intérieur d'une fonction. Elles ne sont pas "visibles" depuis l'extérieur de la fonction. Elles correspondent aux variables muettes vues en mathématiques. Après l'exécution de la fonction, elles sont automatiquement effacées de la mémoire.
- Les variables globales sont extérieures à la fonction. Une fonction ne peut accéder, a priori, qu'en lecture aux variables globales. Si on veut qu'une fonction modifie une variable globale, il faut obligatoirement la déclarer comme étant globale avec la commande :

global noms des variables globales

➪ **Exemple 4 :**

```
def f4(y) :
    x=1
    return y
>>> f4(2)
2
>>> x
Traceback (most recent call last) :
File "<stdin>", line 1, in <module>
NameError : name 'x' is not defined
>>> x=0>>> f4(2)
2
>>> x
0
```

Dans f4, x est une variable locale et n'est donc plus définie après l'exécution de la fonction. Si on défini une variable x global, il n'y a pas de conflit entre les deux variables.

Il est déconseillé de faire apparaître une variable globale dans la définition d'une fonction car la modification de la variable globale après la définition de la fonction pourrait entraîner une modification du comportement de la fonction.

➪ **Exemple 5 :**

```
def f5(x) :
    y+=1
    return x+y
>>> y=1
>>> f5(2)
Traceback (most recent call last) :
File "<stdin>", line 1, in <module>
...
```

L'instruction y+=1 est une modification de la variable y, or seules les variables locales peuvent être modifiées. Ainsi, comme il n'y a pas de variable y définie localement, on obtient un message d'erreur.

```
def f6(x) :
    y=1
    y+=1
    return x+y
>>> f6(2)
4
>>> y
Traceback (most recent call last) :
File "<stdin>", line 1, in <module>
NameError : name 'y' is not defined
```

L'instruction `y+=1` est une modification de la variable `y` et celle-ci a bien été définie comme variable locale, il n'y a pas de message d'erreur lors de l'exécution de `f6(2)`. Par contre, comme `y` est locale, elle est effacée à la fin de l'exécution, donc, on a un message d'erreur si on l'appelle en dehors de l'exécution.

```
def f7(x) :
    global y
    y+=1
    return x+y
>>> y=1
>>> f7(2)
4
>>> f7(3)
6
```

La variable `y` est déclarée comme globale, elle peut donc être modifiée, il n'y a donc pas d'erreur. Mais, il faut faire attention car, lors de l'exécution de `f7(3)`, `y` ne vaut plus 1 mais 2.

```
def f8(x,y) :
    y+=1
    return x+y
>>> y=1
>>> f8(2,y)
4
>>> f8(3,y)
5
```

Ici, `y` est un paramètre de la fonction et pas une variable.